University of Saskatchewan
Department of Computer Science
Cmpt 340
Final Examination

April 19, 2004

Time: 3 hours
Total Marks: 100

Professor: A. J. Kusalik
Closed Book[†]

Name: ______________________________

Student Number: ____________________

Directions:

Answer each of the following questions in the space provided in this exam booklet. If you must continue an answer (e.g. in the extra space on the last page, or on the back side of a page), make sure you clearly indicate that you have done so and where to find the continuation.

Ensure that all answers are written legibly; no marks will be given for answers which cannot be deciphered. Where a discourse or discussion is called for, please be concise and precise. Do not give "extra answers". Extra answers which are incorrect may result in your being docked marks. If you find it necessary to make any other assumptions to answer a question, state the assumption with your answer.

Marks for each major question are given at the beginning of that question. There are a total of 100 marks.

Good luck.

For marking use only:

A.	____/23	D.	____/12
B.	____/9	E.	____/26
C.	____/17	F.	____/13

Total: _____/100

† Closed book, except for one optional 8.5×11 inch quick reference sheet ("cheat sheet") of the student's own compilation.

A. (23 marks)

For each of the following multiple choice[1] questions, indicate the response which best answers or completes the question. Each question is worth **one** mark.

1. Consider the Haskell (HUGS) function:

```
f :: (Float,Float) -> Float
f (x,y) = (a+1) * (b+2)
          where a = (x+y)/2; b = (x+y)/3
```

Variable a can be categorized as

(a) static.

(b) semi-dynamic.

(c) explicitly dynamic.

(d) implicitly dynamic.

(e) global.

(f) None of the above.

2. Recall the tower/1 program from the "Completeness and Correctness" lecture on January 15. That program was

```
% tower(X) is true whenever X is a stack of one or more
% blocks; the bottom block must be on the table.

tower(stack(X,table)) :-
  block(X).

tower(stack(X,Y)) :-
  block(X),
  tower(Y).

% there are N blocks in our world
block(b1).
block(b2).
  .
  .
  .
block(bN).
```

The query

```
?- tower( stack( b1, stack( b1, table ) )
```

succeeds given that program. This means that

(a) $M \supseteq M(P)$

(b) $M(P) \supseteq M$

(c) $M(P) = M$

(d) All of the above.

(e) None of the above

[1] or as Peppermint Patty from the Peanuts comic strip would say, "mystical guess"

3. Recall the `tower/1` program from the "Completeness and Correctness" lecture on January 15. That program was

```
% tower(X) is true whenever X is a stack of one or more
% blocks; the bottom block must be on the table.

tower(stack(X,table)) :-
  block(X).

tower(stack(X,Y)) :-
  block(X),
  tower(Y).

% there are N blocks in our world
block(b1).
block(b2).
  .
  .
  .
block(bN).
```

The query

```
?- tower( stack( b1, stack( b1, table ) )
```

succeeds given that program. This means that

(a) the program is not correct.

(b) the program is not complete.

(c) the program has no logical consequences.

(d) the program is correct and complete.

(e) None of the above.

4. The result of the expression

```
take 3 (from 7) where
   from m = m:from (m+1)
```

is

(a) undefined, ⊥

(b) unknown, because the computation does not terminate

(c) 4

(d) [7,8,9]

(e) none of the above

5. Which of the following functions (assuming they are defined in a programming language with suitable constructs) are referentially transparent?

(a) The factorial function

(b) A function that returns a number input from the keyboard

(c) A function p that counts the number of times it is called

(d) All of the above

6. The single "best" (computer) programming language is:

(a) FORTRAN, since it has been around for so long and is still extensively used.

(b) Prolog, since any knowledge can be expressed in Horn-clause logic.

(c) Context-sensitive grammars since they subsume context-free grammars, and the latter are Turing complete.

(d) C, since it includes disciplined programming constructs (e.g. selective branch, looping, blocking/nesting), yet is expressive enough to allow precise control of machine operations. Further, there is a standard for the language and it is extremely popular (a compiler for the language exists for almost all modern machine architectures).

(e) Java, because it is not only object-oriented and highly-portable, but also because it is a very general purpose language, being used for applications ranging from graphical user interfaces to network programming to parallel programming.

(f) Any language implemented via a pseudo-interpreter.

(g) A language which does not suffer from the "von Neumann bottleneck".

(h) None; there is no such language.

7. Scientific applications (of programming languages) are most accurately characterized by:

(a) extensive use of FORTRAN

(b) large collections of information

(c) elaborate input/output facilities

(d) emphasis on symbol processing

(e) need for access to low-level facilities (e.g. of the operating system)

(f) efficient computation over simple data structures

(g) sophisticated user interfaces

(h) none of the above

8. Which of the following classes of languages would be considered part of the larger class of *declarative languages*?

(a) any language which has a denotational semantics

(b) object-oriented languages

(c) functional languages

(d) any language implemented via a pseudo-interpreter

(e) any language which has a high degree of writeability

(f) none of the above

9. Consider the calculator language implemented in class. What form of scoping does it have?

(a) dynamic scoping.

(b) static (or lexical) scoping.

(c) automatic scoping.

(d) framed scoping.

(e) None of the above, since it didn't implement scoping.

10. Later versions of the "calculator language" developed in class supported functions. What style of parameter passing is provided?

- **(a)** pass by value.
- **(b)** pass by reference.
- **(c)** pass by name.
- **(d)** pass by result.
- **(e)** None of the above.

11. Later versions of the "calculator language" developed in class supported functions. In the implementation, the binding of actual arguments to formal arguments was (is) done at what time?

- **(a)** at program composition time.
- **(b)** at compile time.
- **(c)** at link and load time.
- **(d)** at execution time.
- **(e)** None of the above.

12. In the context of a Prolog program's execution, what is the difference between a (the) proof tree and an (the) AND-tree?

- **(a)** There is no difference; these are synonymous terms.
- **(b)** The proof tree shows what knowledge, in the form of facts, establish the truth of the goal. On the other hand, the AND-tree shows which clauses (and subsequent unifications) were tried in order to find a successful resolution.
- **(c)** The AND-tree is always bigger.
- **(d)** The proof tree captures (shows) more information. For instance, the proof tree indicates the substitutions performed through unification while the AND-tree does not.
- **(e)** None of the above.

13. An "ip-ep pair" is called a

- **(a)** symbol table.
- **(b)** environment.
- **(c)** mini-environment
- **(d)** closure.
- **(e)** None of the above.

14. An object in an object-oriented programming language is an example of what kind of data abstraction?

- **(a)** basic
- **(b)** structured
- **(c)** unit
- **(d)** control
- **(e)** none of the above

15. Which of the following is an example of a control abstraction (in C, Java or other language derived from Algol)?

(a) a function

(b) a conditional statement

(c) an assignment statement

(d) a `goto` statement

(e) All of the above.

16. One approach to overcoming the "von Neumann bottleneck" in current computer architectures is to use:

(a) languages incorporating nondeterminism.

(b) optimizing compilers.

(c) languages with lazy evaluation.

(d) languages with strong typing.

(e) none of the above

17. An object-oriented language is one

(a) in which the keyword "object" appears as a language construct.

(b) in which programs can be conceptualized as consisting of autonomous entities with local state, where the entities communicate with each other in order to perform a computation.

(c) built on the "object calculus" devised by Niklaus Wirth.

(d) which has been derived (descended) from Simula-67, the language which introduced the concept of "object".

(e) None of the above.

18. The main reason that interpreted languages are less efficient than compiled ones is:

(a) An optimizer stage is possible only with compilation.

(b) With an interpreter, decoding of a statement must be performed at runtime and each time the statement is executed.

(c) Interpreters suffer from a phenomenon known as the "von Neumann bottleneck" which compilers can avoid.

(d) In an interpreter, looping is implemented via recursion, while with a compiler it can be implemented using iteration.

(e) None of the above.

19. Which of the following is <u>not</u> a type of optimization that might be performed by an optimizer?

(a) removing redundant register transfers

(b) computing common subexpressions only once

(c) finding and eliminating "dead code"

(d) removing constant operations from loops

(e) removing comments

(f) None of the above.

20. Which of the following languages is (are) often implemented via a pseudo-interpreter (i.e. has a hybrid compiler/interpreter implementation)?

- **(a)** `sh` and `csh` (the language implemented by the *sh* and *csh* command interpreters in UNIX)
- **(b)** Java and Prolog
- **(c)** C and C++
- **(d)** assembly language
- **(e)** All of the above.

21. Another name for an imperative language is a

- **(a)** block-structured language.
- **(b)** statically-scoped language.
- **(c)** procedural language.
- **(d)** applicative language.
- **(e)** All of the above.

22. The symbol "$\perp$" stands for

- **(a)** "bottom" (undefined value)
- **(b)** an anonymous variable
- **(c)** the unification operator
- **(d)** a reduction operator for lambda expressions
- **(e)** a construct which forces normal order evaluation
- **(f)** a Curried function
- **(g)** none of the above

23. The following is an expression (statement) which appears in a program in a particular programming language.

```
f( X, g( Y ) ) = Y
```

Which of the following statements is most likely to be true about this program expression?

- **(a)** It is an example of lazy evaluation.
- **(b)** It is an example of a non-strict function.
- **(c)** It is an example of a goal which will force unification of two terms
- **(d)** It is an example of a unification which will fail due to an occurs-check violation.

B. (9 marks)

Consider the following Haskell/HUGS type signatures:

- **(a)** `[a] -> [a]`
- **(b)** `[Num] -> Num`
- **(c)** `[Num] -> [Num]`
- **(d)** `Num -> Num -> [Num]`

(e) `Eq -> [Eq] -> Bool`

(f) `Eq -> [Eq] -> Int`

(g) `Int -> [a] -> [a]`

(h) `[a] -> [a] -> Bool`

(i) `Int -> [ ( Int -> Int ) ]`

For each function definition below, give the label (e.g. "a", "b", "c") of its type signature from the set above. Assume all functions are Curried.

```
___ -- member indicates whether an element x is a member of a list l
    member x [] = False
    member x (y:l)
        | x == y = True
        | otherwise = member x l

___ -- sum the elements of a list
    sum [] = 0
    sum (n:l) = n + sum l

___ -- return the index of x in list l, or 0 if x does not occur in l
    index x l = index_aux x l 1
      where
        index_aux x [] _ = 0
        index_aux x (y:l) pos
          | x == y = pos
          | otherwise = index_aux x l (pos+1)

___ -- take an argument n and generate a list of length n where the ith
    -- element in the list is a function which adds i to a number it is
    -- given as an argument
    genlist 0 = []
    genlist n = genlist n-1 ++ [ (n +) ]

___ take 0 ls = []
    take n [] = []
    take n (h:t) = h:(take (n-1) t)

___ -- generate a list of numbers from n to m, inclusive
    gennums n m
      | n >= m = []
      | n == m = [n]
      | otherwise = [n] ++ gennums (n+1) m

___ doubleelements [] = []
    doubleelements (n:l) = ((2*n):(doubleelements l))

___ liststructequal [] [] = True
    liststructequal (x:l1) (y:l2) = liststructequal l1 l2
    liststructequal x y = False

___ allbutlast [] = []
    allbutlast [x] = []
    allbutlast (x:l) = (x:allbutlast l)
```

C. (7+3+4+3 = 17 marks)

The following questions require multiple short, precise answers.

1. Consider the following BNF for a portion of the syntax of the calculator language

```
ExpressionSequence ::= sequence ( ExpressionList )

ExpressionList ::= [ ]
ExpressionList ::= [ Expressions ]

Expressions  ::= Expression
Expressions  ::= Expression Expressions
```

(a) What are the terminals in the grammar rules above?

(b) What are the nonterminals in the grammar rules above?

(c) What are the category symbols in the grammar rules above?

(d) What is the name of an alternate grammar formalism based on Prolog?

2. Consider the following Prolog goal

```
?- registration( azx903, program( 2001, philosophy) ) =.. L,
functor( L, F, A ).
```

Will the goal succeed, yes or no? If it does, what will `F` and `A` be bound to?

3. Name four types of information stored in an activation record.

4. Consider the following operator declarations in Prolog:

```
:- op( 900, xfx, gives ).
:- op( 850, fx, concatenating ).
:- op( 800, xfx, and ).
```

With these declarations, a sequence of tokens such as

```
concatenating Xs and Ys gives Zs
```

is parsed as the term

```
gives( concatenating( and( Xs, Ys ) ), Zs )
```

Suppose instead that the operator declarations had been

```
:- op( 850, xfx, gives ).
:- op( 900, fx, concatenating ).
:- op( 800, xfy, and ).
```

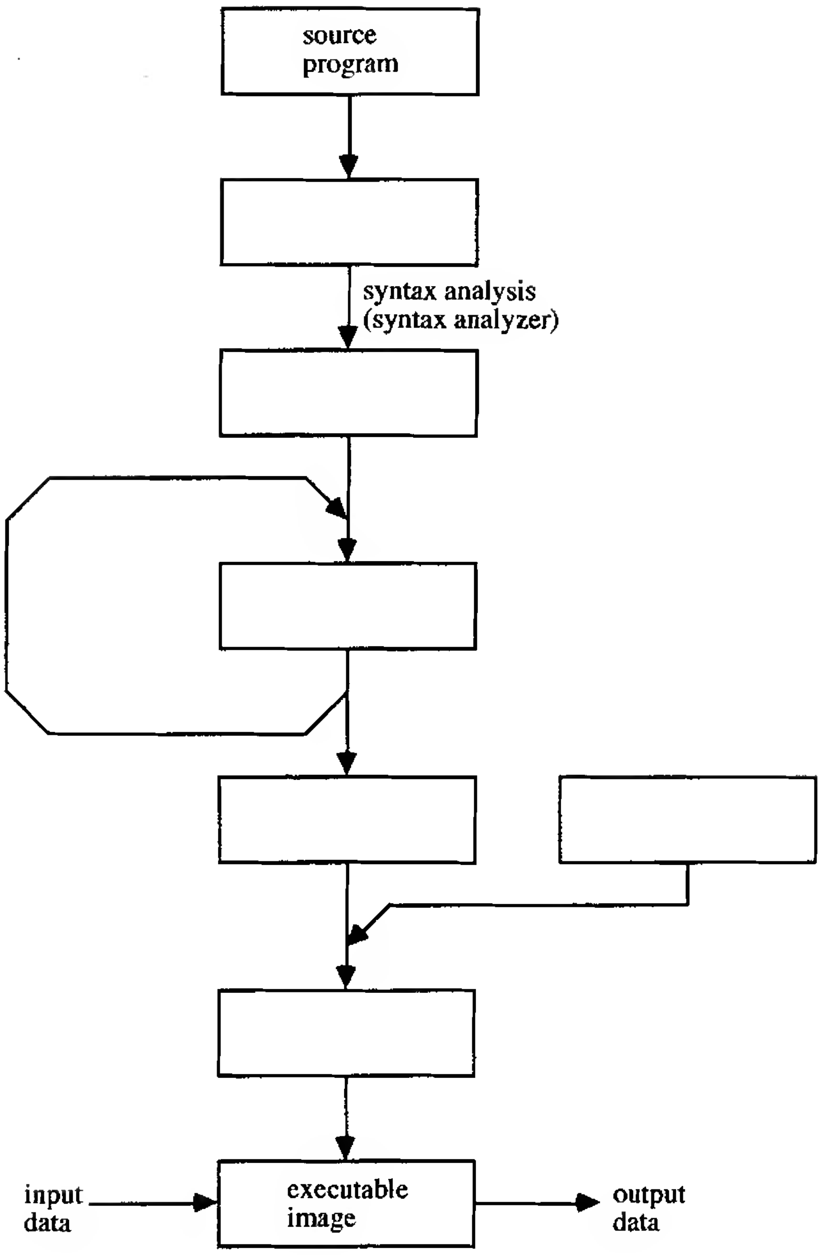
What term would be the result of parsing the following sequence of tokens

```
concatenating Xs and Ys and Zs gives L
```

given the last set of operator declarations? If you wish, you can specify the term (your answer) in the form of a tree.

D. (6+6 = 12 marks)

On the next page is a (partial) diagram of the steps involved in a language implementation by way of compilation. Rectangles represent types of information, and arcs (directed line segments) represent transformations or processing stages which transform one type of information to another. Label each rectangle in the diagram with the type of information at that stage. Label <u>each</u> arc with the type of process (or processing stage) it represents. A few labels are already provided for you to get you started.



E. (5+3+3+5+3+4+3 = 26 marks)

Answer each of the following questions with a discussion-oriented answer. Where appropriate, use examples and diagrams to illustrate your points.

1. Give the name of a looping construct (method to achieve repeated execution of a portion of code) that is possible in Prolog and logic programming languages, but not in most other programming languages paradigms. Then give a realistic example of use of this construct. The example must be explained and must include program code.

2. In class, the interpreter for the calculator language was written in Prolog. What were some of the advantages of implementing the interpreter this way over, say, implementing it in C or Java? Give at least 3 advantages.

3. Given only the following Prolog clause in Prolog's databse:

```
human( bob ).
```

Prolog will respond as follows:

```
?- human( X ).
X = bob

?- not( human( X ) ).
no
```

Why did Prolog respond to the last goal as it did? Is Prolog saying that there is no X such that X is not human? Explain your answer.

Note: not/1 is typically defined to be synonymous with the operator '\+' in most Prolog systems.

4. The term "environment" arose several times in the course material, and with a different meaning each time. Pick three such meanings for the term "environment". Explain and distinguish (differentiate) each meaning.

5. In the context of programming languages, what is "the heap"? Where is it? What is stored there?

6. A number of programming language design criteria can be considered to be facets of an overall principle of efficiency. Give two such criteria, and explain how they can be understood as aspects of (some sort of) efficiency.

7. Explain what Currying is, and how its presence in a functional language is advantageous. Use a good example in your explanation.

F. (5+8 = 13 marks)

A common operation in computer science is to form the intersection of sets. Assume a representation of sets as lists. Assume a list contains no duplicate elements; i.e. the lists will contain only unique elements. Write programs in each of the languages below to implement the intersection of two sets (lists) and provide the resulting set as a list.

Prolog: In the case of Prolog, call the predicate `intersection/3` and assume that it will be called with its first two arguments completely instantiated (ground lists). Use '`->;`' rather than `not/1` or red cuts. You can assume that `member/2` and `append/3` are predefined.

Haskell/HUGS: In the case of Haskell/HUGS, assume that all elements of the sets (lists) are of the same type. Also, provide explicit type signatures.

Epilog

That's it! You're all done!